Análise Léxica I

Eduardo Ferreira dos Santos

Ciência da Computação
Centro Universitário de Brasília – UniCEUB

Março, 2017

Sumário

UniCEUB
Centro Universitário de Brasília

Fases

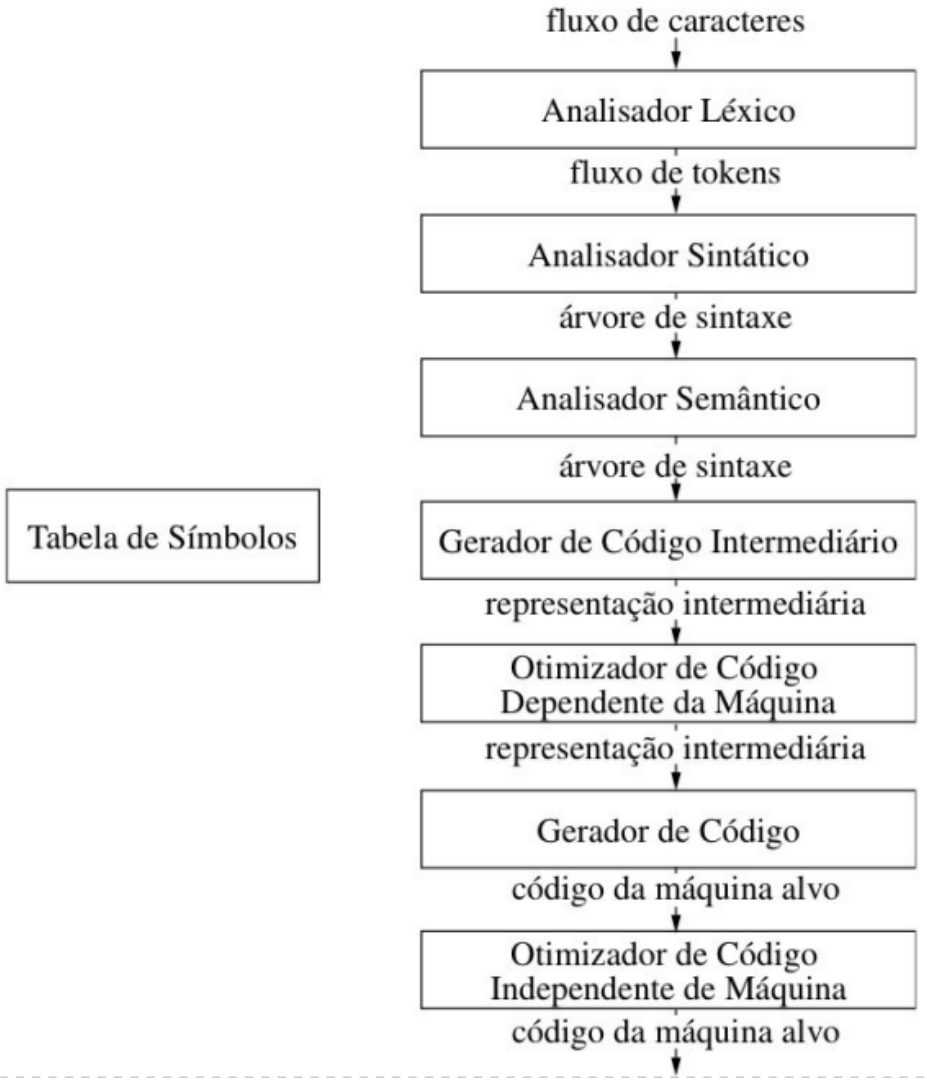

Figura 1.1: Fases do compilador [Aho et al., 2007]

Execução

- A fase de análise divide o programa e impõe uma estrutura gramatical;
- Se houver algum erro, deve fornecer mensagens esclarecedoras;
- A fase de síntese constrói o programa objeto usando:
  - Representação intermediária;
  - Tabela de símbolos.
- Normalmente a compilação é dividida em fases, como descrito na Figura 4.

Análise léxica

- Normalmente a compilação se inicia pela análise léxica;
- Analisador léxico: lê o fluxo de caracteres e agrupa em sequências significativas;
- As sequências significativas são chamadas lexemas;
- Para cada lexema, o analisador léxico produz como saída um token;

$$\langle nome\_token, valor\_atributo \rangle \tag{1}$$

- O token é enviado para a etapa seguinte, a análise sintática.

Exemplo

$$position = initial + rate * 60 \tag{2}$$

position Mapeado para o token $\langle id, 1 \rangle$:

- id Símbolo abstrato que significa identificador;
- 1 Entrada na tabela de símbolos onde está *position*.

= Mapeado para o token $\langle = \rangle$. Por não exigir um valor de atributo, omitimos o segundo componente;

initial Mapeado para o token $\langle id, 2 \rangle$;

+ Mapeado para o token $\langle + \rangle$;

rate Mapeado para o token $\langle id, 3 \rangle$;

* Mapeado para o token $\langle * \rangle$;

60 Mapeado para o token $\langle 60 \rangle$[1].

[1] Para o lexema 60 deveria haver um token como $\langle numero, 4 \rangle$

Atribuição

- Após a análise léxica, executa-se o comando de atribuição;
- O comando gera as substituições apontadas no exemplo 2;
- Após a substituição, obtemos o resultado 3:

$$\langle id, 1\rangle\langle =\rangle\langle id, 2\rangle\langle +\rangle\langle id, 3\rangle\langle *\rangle\langle 60\rangle \tag{3}$$

- É possível perceber que alguns tokens são símbolos abstratos;
- Os símbolos representam operadores.

Análise sintática

- Analisador sintático: utiliza os primeiros componentes dos tokens para gerar uma representação de árvore;
- A árvore representa a estrutura gramatical dos tokens;
- Representação da árvore de sintaxe:
  - Cada nó interior representa uma operação;
  - Filhos dos nós representam os argumentos da operação.
- As próximas fases do compilador utilizam a estrutura gramatical;
- Gerar o programa fonte;
- Gerar o programa objeto.

Árvore de sintaxe

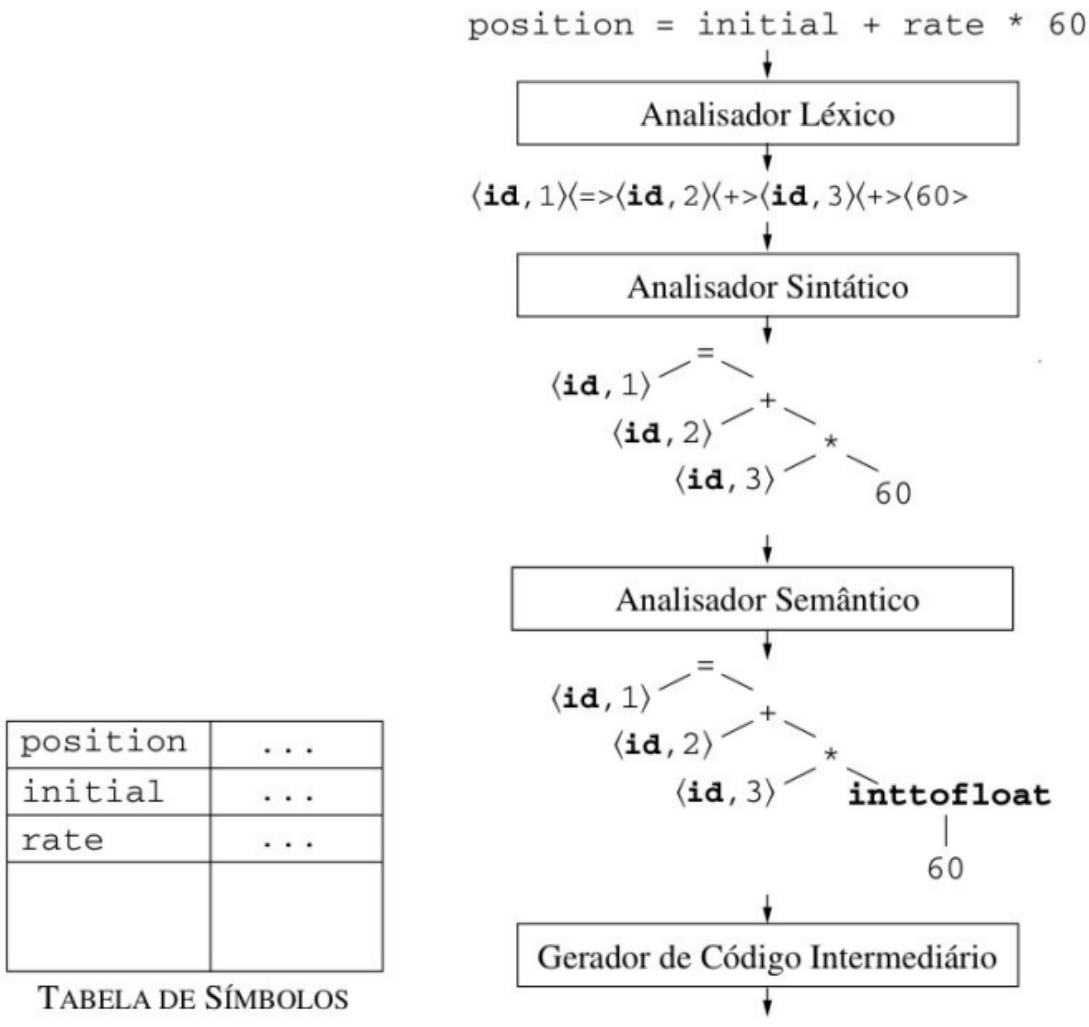

Figura 1.2: Tradução de uma instrução para o exemplo 2 [Amarasinghe and Rinard, 2010]

Papel do analisador

- Primeira fase do compilador;
- Tarefa principal [Aho et al., 2007]:
  1. *Ler os caracteres de entrada do programa fonte;*
  2. *Agrupá-los em lexemas;*
  3. *Produzir uma sequência de tokens para cada lexema.*
- Envia o fluxo de tokens para o analisador sintático;
- Ao encontrar um identificador insere o lexema na tabela de símbolos.

Interação léxico sintático

- Em alguns casos, o analisador léxico precisa ler a tabela de símbolos para encontrar o token apropriado para enviar ao analisador sintático;
- Normalmente o analisar sintático chamada o analisador léxico;
- Lê os caracteres de entrada até identificar o próximo lexema, produzindo o próximo token;
- Chamada *getNextToken*;
- Retorna ao analisador sintático.

Interações

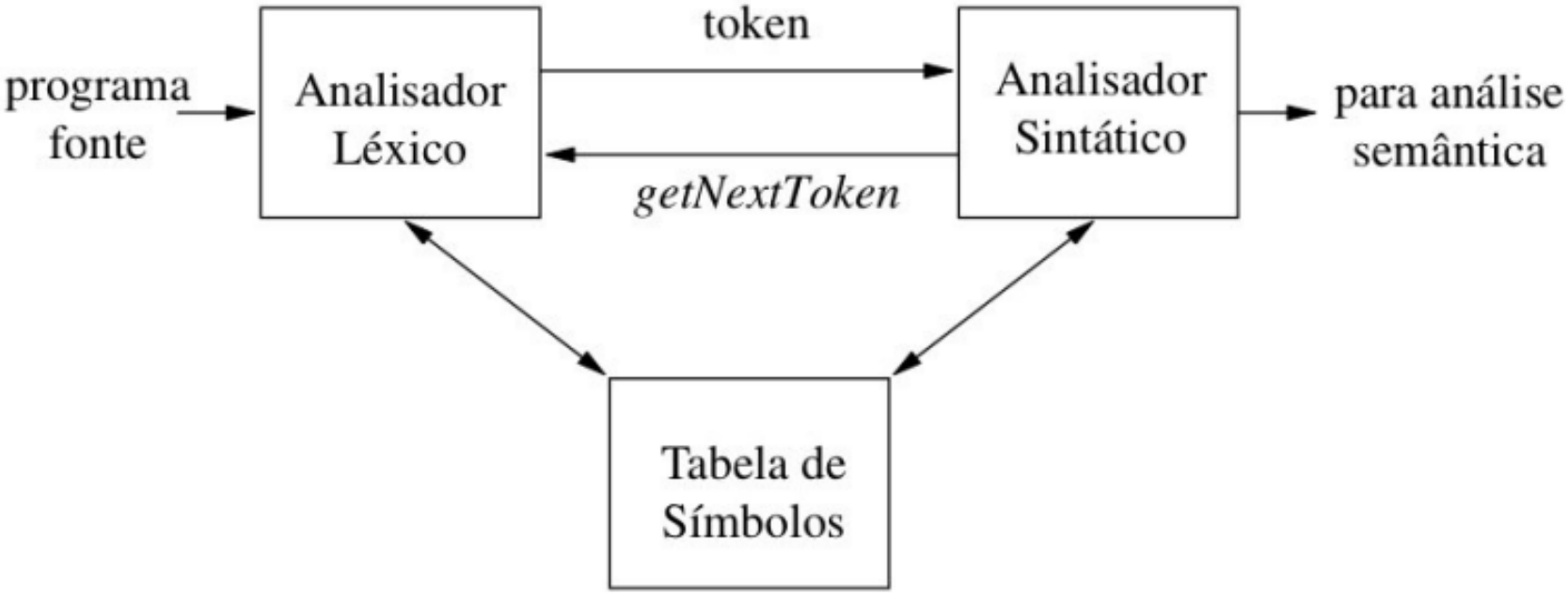

Figura 2.1: Interações entre o analisador léxico e o analisador sintático [Aho et al., 2007]

Outras tarefas

- Outras tarefas que podem ser realizadas pelo analisador léxico:
  - Remover espaços em branco;
  - Correlacionar mensagens de erro com o programa fonte;
    - Registra o número de quebras de linha;
    - Associa a mensagem de erro ao número da linha;
    - Pode inclusive fazer uma cópia do programa fonte com as mensagens de erro.
  - Expansão de macros;
- Divisão em cascata de dois processos:
  a) Escandimento;
  b) Análise léxica.

Léxica versus Sintática

- Por que separar análise léxica de sintática?

  Simplicidade Diminui a complexidade de uma das tarefas:
  - Ignorar espaços em branco não é uma unidade sintática;
  - Pode gerar uma projeto de linguagem mais limpo.

  Eficiência Aumenta a eficiência do compilador:
  - Técnicas que executam apenas a tarefa léxica;
  - Utilização de técnicas de *buffering*.

  Portabilidade Melhora a portabilidade do compilador:
  - Características específicas do dispositivo de entrada podem ser restringidas ao analisador léxico.

Tokens, padrões e lexemas

Token Dupla formada pelo nome e valor de atributo (opcional)

- Nome do token: símbolo abstrato que representa um tipo de unidade léxica [Aho et al., 2007];
- Também é um símbolo de entrada para o analisador sintático.

Padrões Descrição da forma que os lexemas de um token podem assumir:

- Palavra-chave como token;
- Casamento de sequências de caracteres.

Lexemas Sequência de caracteres no programa fonte que casa com o padrão para um token.

- Identificado como instância desse token.

Exemplo em C

Listing 1: Exemplo para a linguagem C [Aho et al., 2007]

```
printf("Total = %d\n", score);
```

- Tanto `printf` quanto `score` são casados com o o padrão para o token **id**;
- O trecho "`Total = %d{*n"}` é um lexema casando com um **literal**.

Exemplo de tokens

TOKEN	DESCRIÇÃO INFORMAL	EXEMPLOS DE LEXEMAS
if	caracteres `i`, `f`	`if`
else	caracteres `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` ou `<=` ou `>=` ou `==` ou `!=`	`<=`, `!=`
id	letra seguida por letras e dígitos	`pi`, `score`, `D2`
number	qualquer constante numérica	`3.14159`, `0`, `6.02e23`
literal	qualquer caractere diferente de ", cercado por "s	`"core dumped"`

Figura 2.2: Exemplos de tokens [Aho et al., 2007]

Regras de tokens

1. Um token para cada palavra-chave. O padrão para a palavra-chave e ela mesma;
2. Tokens para os operadores, seja individualmente ou em classes;
3. Um token para todos os identificadores;
4. Um ou mais tokens representando constantes;
5. Tokens para cada simbolo de pontuação.

Atributos de tokens

- É possível mais de um lexema casar com o padrão;
- Ex.: Token `number` casa com **0 e 1**;
- O analisador léxico pode passar para o sintático o nome do token e um valor de atributo que descreve o lexema;
- O nome do token influencia as decisões da análise sintática;
- O valor to token influencia a tradução dos tokens;
- O valor de atributo para um identificador aponta para sua entrada na tabela de símbolos.

Exemplo para Fortran

- Consideremos a seguinte expressão em Fortran:

```
E = M * C ** 2
```

- A instrução pode ser descrita como uma sequência de pares:

⟨ **id**, apontador para a entrada da tabela de símbolos de E ⟩
⟨ assign_op ⟩
⟨ **id**, apontador para a entrada da tabela de símbolos de M ⟩
⟨ mult_op ⟩
⟨ **id**, apontador para a entrada da tabela de símbolos de C ⟩
⟨ exp_op ⟩
⟨ **number**, valor inteiro 2 ⟩

Erros léxicos

```
fi (a == f(x)) ...
```

- O identificador `fi` é:
  - A palavra-chave `if` escrita errada?
  - Identificador de função não declarado?
- Sendo `fi` um lexema válido para o token **id**, o tratamento deve acontecer em outra fase do compilador.

Pares de buffer

- Programas grandes podem ter muitos caracteres a processar;
- Utilização de técnicas de *buffering*:

  lexemeBegin Início do lexema corrente;

  forward Lê adiante até que haja casamento de padrão;

  eof Marca o fim do arquivo fonte.

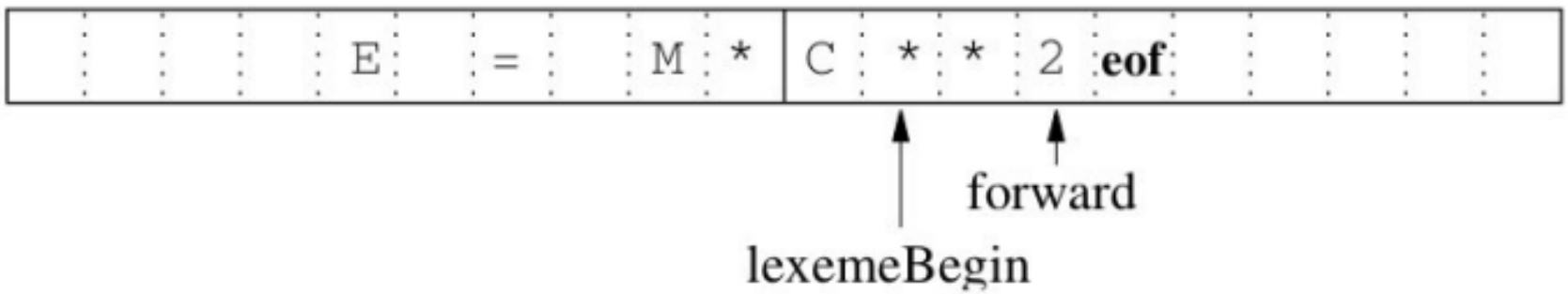

Figura 2.3: Usando um par de buffer de entrada [Aho et al., 2007]

Algoritmo de buffer

1. Ao encontrar o próximo lexema, `forward` é configurado para apontar o último caractere à direita;
2. Registra o lexema como valor de um atributo de um token;
3. Retorna para o analisador sintático;
4. `lexemeBegin` aponta para o próximo caractere.

Sentinelas no buffer

- Identifica a necessidade de recarregar um dos *buffers*;
- Teste do fim do *buffer* com teste do caractere corrente;
- Utilização de sentinela: caractere de fim de arquivo (**eof**).

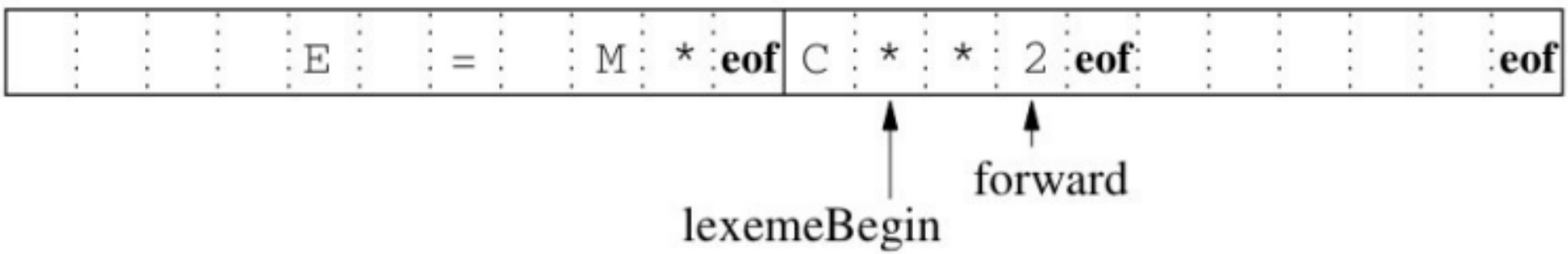

Figura 2.4: Sentinelas no fim de cada buffer [Aho et al., 2007]

Sentinelas

Listing 2: Código de lookahead com sentinelas [Aho et al., 2007]

```
switch (*forward++) {
  case eof:
    if (forward áest no fim do primeiro buffer) {
      recarrega segundo buffer;
      forward = íincio do segundo buffer;
    } else if (forward áest no fim do segundo buffer) {
      recarrega primeiro buffer;
      forward = íincio do primeiro buffer;
    } else {
      /* eof dentro de um buffer marca o fim da entrada
          */
      termina áanlise élxica
    }
    break;
  Casos para outros caracteres
}
```

Exercício 1

Listing 3: Exemplo do exercício 01 [Aho et al., 2007]

```
float limitedSquare(x) float x {
  /* retorna x ao quadrado, mas nunca mais do que 100 */
  return (x <= -10.0 || x >= 10.0) ? 100 : x*x;
}
```

1. Divida o programa escrito em C++ nos lexemas apropriados;
2. Quais lexemas devem ter valores léxicos associados?
3. Quais são esses valores?

Exercício 2

Listing 4: Exemplo do exercício 02 [Aho et al., 2007]

```
Here is a photo of <b>my house</b>:
<p><img src = "house.gif"><br>
Veja <a href = "morePix.html">More Pictures</a> if you
liked that one</p>
```

1. Divida o documento HTML nos lexemas apropriados;
2. Quais lexemas devem ter valores léxicos associados?
3. Quais são esses valores?

OBRIGADO!!!
PERGUNTAS???


Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007).
Compiladores–Princıpios Técnicas e Ferramentas.
Pearson, 2a. edition.


Amarasinghe, S. and Rinard, M. (2010).
Computer language engineering.
Disponível em `http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-035-computer-language-engineering-spring-2010/` Acessado em 02/08/2016.